NPL Search Results

21/5/1 (Item 1 from file: 8)
DIALOG(R)File 8: Ei Compendex(R)

MULTITASKING-PASCAL EXTENSIONS SOLVE CONCURRENCY PROBLEMS.
Mackie, Peter H.
Corresp. Author/ Affil: Mackie, Peter H.
EDN (EDN) 1982 27/19 (145-149)
Publication Date: 19820101
Document Type: Trade Journal **Record Type:** Abstract
Language: Unspecified **Summary Language:** English

To avoid deadlock (one process **waiting** for a **resource** that **another process** cannot release) and indefinite postponement (**one process** being **continually** denied a **resource** request) in multitasking-system application, a high-level development language can be used with built-in concurrency handlers. Parallel PASCAL is one such language; it extends standard PASCAL via special task synchronizers: a new data type called Signal, new system procedures called Wait and Send and a Boolean function termed Awaited. The problems the multitasking PASCAL helps to solve are examined in order to understand the use of the language.

21/5/2 (Item 1 from file: 60)
DIALOG(R)File 60: ANTE: Abstracts in New Tech & Engineer

Method and apparatus for synchronizing threads on a processor that supports transactional memory
Chaudhry, Shailender; Tremblay, Marc; Caprioli, Paul , USA
Document Type: Patent **Record Type:** Abstract
Language: English

One embodiment of the present invention provides a system that synchronizes threads on a multi-threaded processor. The system starts by executing instructions from a multi-threaded program using a first thread and a second thread. When the first thread reaches a predetermined location in the multi-threaded program, the first thread executes a Start-Transactional-Execution (STE) instruction to commence transactional execution, wherein the STE instruction specifies a location to branch to if transactional execution fails. During the subsequent transactional execution, the first thread accesses a mailbox location in **memory** (which is also accessible by the second thread) and then executes instructions that cause the first thread to **wait**. When the **second thread** reaches a **second** predetermined location in the multi-threaded program, the second thread signals the first thread by accessing the mailbox location, which causes the transactional execution of the first thread to fail, thereby causing the **first thread** to **resume** non-transactional execution from the location specified in the STE instruction. In this way, the second thread can signal to the first thread without the first thread having to poll a shared variable.

27/5/1 (Item 1 from file: 8)
DIALOG(R)File 8: Ei Compendex(R)

Breakpoints and halting in distributed programs .
Miller, Barton P.; Choi, Jong-Deok
Corresp. Author/ Affil: Miller, Barton P.: Univ of Wisconsin, Madison, WI, USA
Conference Title: Proceedings - 8th International Conference on Distributed Computing Systems.
Conference Location: San Jose, CA, USA
Sponsor: IEEE, Computer Soc, Technical Committee on Distributed Processing; IEEE, New York, NY, USA

Proceedings - International Conference on Distributed Computing Systems (Proc Int Conf Distrib Comput Syst) 1988 8/- (316-323)
Publication Date: 19881201
Publisher: Publ by IEEE
Document Type: Conference Paper; Conference Proceeding **Record Type:** Abstract
Language: English **Summary Language:** English
Number of References: 12

Interactive debugging requires that the programmer be able to **halt** a **program** at interesting **points** in its **execution**. The authors define distributed **breakpoints** and present an algorithm for implementing the detection **points** and an algorithm for **halting** a distributed **program** in a consistent **state**. **Events** that can be partially ordered are defined as detectable and form the basis for the **breakpoint** predicates. From the **breakpoint** definition, an algorithm is obtained that can be used in a distributed debugger to **detect** these **breakpoints**. The **halting** algorithm extends K.M. Chandy and L. Lamport's (1985) algorithm for recording global **state** and solves the problem of **processes** that are not fully connected or frequently communicating.

27/5/2 (Item 1 from file: 35)
DIALOG(R)File 35: Dissertation Abs Online

CHANGING TASK DEMANDS IN SUSTAINED ATTENTION: EFFECTS ON PERFORMANCE AND PERCEIVED WORKLOAD (VIGILANCE)
Author: GLUCKMAN, JONATHAN PETER **Degree:** PH.D.
Year: 1990
Corporate Source/Institution: UNIVERSITY OF CINCINNATI (0045)
Source: Volume 5111B of Dissertations Abstracts International.
PAGE 5614 . 137 PAGES

This investigation assessed the effects of shifts in **task** demand upon vigilance performance and perceived mental workload. The problem was attacked experimentally by shifting subjects from single- to dual- **task** monitoring, or vice-versa, midway through a 40-min vigil. In most cases perceptual sensitivity (d$\sp\prime$) for shifted groups during the post-shift phase did not differ significantly from their dual- or single-**continuous task** controls, respectively. The sole **exception** to this pattern was a transient effect in which the performance of subjects shifted from dual- to single-**task** was poorer than non-shifted controls immediately after the shift, but improved to an equivalent level by the **end** of the vigil. The results were explained in terms of a multidimensional view of **resource** theory which suggested that performance efficiency is related to the ability of the processing system to expand **resources** sufficiently to cope with **task** demands, and that **resources** are consumed over time. It was also suggested that a compensatory regeneration mechanism, analogous to that described in regard to exercise physiology, could account for recovery from the transient effect noted earlier. From an applied perspective, the results suggest that when initial monitoring demands are maintained at a low level, few carryover effects on performance will be seen when demand oscillations occur in the form of increases in the number of monitoring **tasks**. When initial demand is high (dual-**task**), temporary carryover effects will occur when demand is reduced.

Workload scores fell uniformly in the upper third of the NASA-TLX scale, and did not differ among the experimental groups. Mental Demand and Frustration were found to be the major contributors to workload. The results of the workload analysis provide strong evidence against theoretical positions which hold that vigilance is understimulating or underarousing. Instead, they suggest that such **tasks** are quite demanding, and that **resource** theory, which focuses on **task** -related drains in mental processing capability, affords a better vehicle by which to account for vigilant behavior.

27/5/3 (Item 1 from file: 2)
DIALOG(R)File 2: INSPEC

Supporting reverse execution of parallel programs
Author(s): Pan, D.Z.[1]; Linton, M.A.[1]
Affiliation(s):

[1] Stanford Univ., CA, USA
Journal: SIGPLAN Notices , vol.24 , no.1 , pp.124-9
Country of Publication: USA
Publication Date: Jan. 1989
Conference Title: ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging
Conference Date: 5-6 May 1988
Conference Location: Madison, WI, USA
Conference Sponsor: ACM
Language: English
Document Type: Conference Paper in Journal (PA)

Parallel programs are difficult to debug because they run for a long time and two executions may yield different results. Reverse execution is a simple and powerful concept that solves both these problems. The authors are designing a tool for debugging parallel programs, called Recap, that provides the illustration of reverse execution using checkpoints and event recording and playback. During normal execution, Recap logs the results of system calls and shared **memory** reads, as well as the times that asynchronous events (signals) occur. Recap periodically checkpoints the state of a **process** by forking and **suspending** a new **process**. To reverse execute to a certain point in time, Recap **continues** the nearest checkpoint **process** forward in a self-contained environment, simulating all events using the log. The authors are implementing Recap as part of a larger environment for parallel program development. (*8 refs.*)

27/5/4 (Item 2 from file: 2)
DIALOG(R)File 2: INSPEC

Hot back-up and fault tolerant controllers
Author(s): Leach, A.[1]
Affiliation(s):
[1] Texas Instrum. Ltd., Bedford, UK
Journal: Control & Instrumentation , vol.19 , no.1 , pp.43-4
Country of Publication: UK
Publication Date: Jan. 1987
Language: English
Document Type: Journal Paper (JP)

As control systems become ever more powerful and able to handle the most demanding control requirements, the dependability of these units and their ability to tolerate faults and maintain normal operation has become a subject of keen interest. Nowhere has this been more evident than in the **process** industries where the cost of **halting** and then restarting production is generally extremely high, and the time lost in recovering from any **stoppage**, whether caused inadvertently or requested intentionally, is extremely high. In extreme cases, the investment in additional **equipment** which provides back-up can be completely saved on the first occasion that its use becomes necessary. With the **exception** of distributed control systems designed for **continuous processes**, the availability of hot back-up systems is a fairly recent innovation, as is the advent of fault tolerant control systems. So, the level of understanding of the capabilities of the various solutions currently on offer, and how they meet the differing needs of each application, is currently low. There is thus a need for education into, the analysis of application requirements for back-up and the implications of installing such systems and maintaining them in an operational state with full back-up facilities available after installation. (*0 refs.*)

27/5/7 (Item 1 from file: 60)
DIALOG(R)File 60: ANTE: Abstracts in New Tech & Engineer

Parallel and asynchronous debugger and debugging method for multi-threaded programs
Sistare, Steven J; Plauger, David , USA
Document Type: Patent **Record Type:** Abstract

Language: English

A debugger for aiding in the debugging of multi-threaded program, in response to an event such as, for example, a **breakpoint** in a **thread** which has caused an operating system to **stop** execution of all **threads**, identifies the **thread** which contained the **breakpoint**. After identifying the **thread** which contained the **breakpoint**, the debugger enables the operating system to **resume** execution of the **other threads**, that is, the **threads** which did not contain the **breakpoint**. By allowing the **other threads**, that is, the **threads** which did not contain the **breakpoint**, to continue execution, the debugger's impact on program execution is substantially reduced, particularly for programs which contain a large number of **threads**.

27/5/8 (Item 2 from file: 60)
DIALOG(R)File 60: ANTE: Abstracts in New Tech & Engineer

Debugger apparatus and method for indicating time-correlated position of threads in a multi-threaded computer program
Bates, Cary Lee; Santosuosso, John Matthew , USA
Document Type: Patent **Record Type:** Abstract
Language: English

A debugger inserts instrumentation hooks in a multi-threaded computer program that allow collecting a program trace and that provide timestamps that correspond to the program trace. When a **breakpoint** in a first **thread** is encountered, a timestamp corresponding to the **breakpoint** is retrieved. Execution of the **other threads** may **continue** until the debugger is able to **halt** their execution. Once the execution of all **threads** has been **halted**, the program trace for each **thread** is traced backwards to a point where the timestamp is less than the **breakpoint** timestamp. Instructions are then executed, one by one, until the execution time of the instructions plus the timestamp is approximately the same as the **breakpoint** timestamp. The instruction in the program trace display is then highlighted to indicate the instruction that was likely being executed when the **breakpoint** in the first **thread** is encountered.

27/5/9 (Item 3 from file: 60)
DIALOG(R)File 60: ANTE: Abstracts in New Tech & Engineer

Method and system for intelligent and adaptive exception handling
Wood, Eric; Tsyganskiy, Boris , USA
Publisher Url: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&u=/netahtml/PTO/search-adv.htm&r=1&p=1&f=G&l=50&d=PTXT&S1=7543175.PN.&OS=pn/7543175&RS=PN/7543175
Document Type: Patent **Record Type:** Abstract
Language: English

A method and system for handling errors and **exceptions** in an ERP environment are disclosed. According to one aspect of the present invention, a condition or event causes a script-engine associated with a particular ERP server to generate an error message. The error message is communicated to a centralized controller-node. The centralized controller-node analyzes the message and determines the best course of action for handling an error or **exception** related to the error message. Based on the controller node's analysis, the controller node communicates a response message, thereby enabling the **process** that caused the error to **continue** without **terminating** abnormally.

27/5/10 (Item 4 from file: 60)
DIALOG(R)File 60: ANTE: Abstracts in New Tech & Engineer

System and method for creating a restartable non-native language routine execution environment
Abelite, Margaret A; Cotner, Curt Lee , USA
Publisher Url: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&u=/netahtml/PTO/search-adv.htm&r=1&p=1&f=G&l=50&d=PTXT&S1=7478387.PN.&OS=pn/7478387&RS=PN/7478387
Document Type: Patent **Record Type:** Abstract
Language: English

A computer program product is provided as a system and associated method to provide a robust, fail-safe, fast environment for running non-native language routine programs, such that when the program fails or **terminates**, the virtual machine can be **restarted** without **terminating** the operating system **process**. The system dynamically creates a number of independent virtual machines to run non-native language routines. Each virtual machine is created in a **process** external to the database engine, in a refreshable native runtime environment inside the **process** or address space. If the virtual machine should **terminate** with an uncaught **exception** or error or if it becomes unusable for executing other non-native language routines, the system recognizes that condition and **terminates** the native runtime environment without **terminating** the **process**. When a failed non-native language routine is **terminated**, the system cleans up any **threads** related to the **termination** of the virtual machine, freeing **memory** allocated to the **threads** and eliminating any possible conflict between old **threads** and the **next** non-native language routine operated by the virtual machine.

27/5/12 (Item 6 from file: 60)
DIALOG(R)File 60: ANTE: Abstracts in New Tech & Engineer

Cross address space thread control in a multithreaded environment
Ault, Donald F; Bender, Ernest S; Franks, Jon K; Walkowiak, Steven , USA
Publisher Url: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&u=/netahtml/PTO/search-adv.htm&r=1&p=1&f=G&l=50&d=PTXT&S1=5632032.PN.&OS=pn/5632032&RS=PN/5632032
Document Type: Patent **Record Type:** Abstract
Language: English

A method of controlling the execution of the **threads** of a first application such as a user application from a second application such as a debugger application running in a different address space. After initializing trace mode for the user application, the debugger **waits** for an event to occur on one of the **threads** of the user application. Upon the occurrence of an event on one of the user application **threads**, an event handler obtains control of the **thread execution**. The **event** handler **suspends** execution of the remaining **threads** in the application, posts the debugger and then **suspends** its own execution. When the debugger application has completed its debugging operations, it posts the event handler, which **resumes** execution of the **suspended threads** and returns control to the **thread** on which the event occurred. If a subsequent event occurs on one **thread** while a previous event on **another thread** is being processed, the event handler for the subsequent event places it in a deferred event queue for deferred processing. Events consisting of **breakpoints** are redriven rather than being placed on the deferred queue. The debugger application may hold selected **threads** in a **suspended** state following **resumption** of the remaining **threads** by setting hold flags associated with those **threads**.

39/5/1 (Item 1 from file: 8)
DIALOG(R)File 8: Ei Compendex(R)

Disk arm movement in anticipation of future requests
King, Richard P.
Corresp. Author/ Affil: King, Richard P.: **IBM** Thomas J. Watson Research Cent, Yorktown Heights, United States
ACM Transactions on Computer Systems (ACM Trans Comput Syst) 1990 8/3 (214-229)

Publication Date: 19901201
Item Identifier (DOI): 10.1145/99926.99930
Document Type: Article; Journal **Record Type:** Abstract
Language: English **Summary Language:** English
Number of References: 29

When a disk drive's access arm is **idle**, it may not be at the ideal location. In anticipation of future requests, movement to some other location may be advantageous. The effectiveness of anticipatory disk arm movement is explored. Various operating conditions are considered, and the reduction in seek distances and request response times is determined for them. Suppose that successive requests are independent and uniformly distributed. By bringing the arm to the middle of its range of motion when it is **idle**, the expected seek distance can be reduced by 25 percent. Nonlinearity in time versus distance can whittle that 25 percent reduction down to a 13 percent reduction in seek time. Nonuniformity in request location, nonPoisson arrival **processes**, and high arrival rates can whittle the reduction down to nothing. However, techniques are discussed that maximize those savings that are still possible under those circumstances. Various systems with multiple arms are analyzed. Usually, it is best to spread out the arms over the disk area. The **exception** is duplexed disk systems with more than half of the requests being for writing. Then both arms should be brought to the middle.

39/5/4 (Item 2 from file: 2)
DIALOG(R)File 2: INSPEC

Linking program modules on a stack microprocessor
Author(s): Lang, D.J.[1]; Luiz, F.A.[1]; Peters, T.C.[1]; White, C.P.[1]
Affiliation(s):
[1] **IBM**. Armonk, NY, USA
Journal: IBM Technical Disclosure Bulletin , vol.19 , no.12 , pp.4528
Country of Publication: USA
Publication Date: May 1977
Language: English
Document Type: Journal Paper (JP)

This invention relates particularly to a method for directly or indirectly linking modules, whose order of execution is determined by a push-down **process** stack, and for verifying whether a target address contains the desired entry point into the module. The method comprises the steps of **suspending** execution of the current module (including locking out interrupts) and saving of the status and parameters necessary for re-entry; retrieving the target address either directly or indirectly; fetching the instruction specified by the address; verifying whether the instruction defines the desired entry point (an ` Enter' instruction); if verified, completing the initialization of the top of the **process** stack; and if not verified, posting a **program check**. The execution of the Enter instruction from the new object module completes the conversion of the **process** stack. (*0 refs.*)

39/5/5 (Item 3 from file: 2)
DIALOG(R)File 2: INSPEC

Technique for nondisruptive quiesce and replace of procedures [in multiprogramming]
Author(s): Gilbert, D.C.[1]; Taradalsky, M.[1]
Affiliation(s):
[1] **IBM**. Armonk, NY, USA
Journal: IBM Technical Disclosure Bulletin , vol.18 , no.6 , pp.1937-41
Country of Publication: USA
Publication Date: Nov. 1975
Language: English
Document Type: Journal Paper (JP)

To **quiesce** a procedure is to cause all **processes** which call that procedure to go into a **wait** state.

The state of any **process** already executing the procedure at the time is not changed. The disclosure shows how this operation can replace a certain procedure by another for all calls after a certain time, without causing **abnormal termination** of any **process**. (*0 refs.*)

39/5/7 (Item 1 from file: 95)
DIALOG(R)File 95: TEMA-TECHNOLOGY & MANAGEMENT
. All rights reserved.

Dynamic synchronization of real-time threads for multiprocessor systems
Hongyi Zhou; Schwan, K; Gheith, A
Georgia Inst. of Technology, Atlanta, USA; **IBM** Austin, USA
SEDMS 3, Symposium on Experiences with Distributed and Multiprocessor Systems, USENIX, Newport Beach, CA, USA, March 26 - 27, 1992 , 1992
Document type: Conference paper **Language:** English
Record type: Abstract

The authors study mutual exclusion and synchronization for dynamic hard real-time multiprocessor applications. As with any dynamic parallel program, a dynamic real-time application's execution can result in on-line creation of additional **tasks**, and the creation of such time-constrained **tasks** cannot be predicted or accounted for prior to program execution. The research results presented in this paper concern **task** synchronization such that guarantees can be made without performing on-line schedulability analysis and on-line analysis concerning the maximum time that a **task** will **wait** for some resource being acquired with a synchronization primitive. The authors present a real-time locking scheme that prevents deadlocks and ensures time-bounded mutual exclusion. The maximum **waiting** time for a **task** attempting to acquire a resource is computed with an O(1) algorithm. Two important attributes of the algorithm are: (1) previously made guarantees regarding resource accesses are always maintained, and (2) failures regarding accesses to shared resources are reported immediately. As a result, the application program or higher-level operating system software can deal with such failures in a timely manner, by acquisition of alternative resources, by execution of **exception** handling code, etc.

20/3,K/1 (Item 1 from file: 275)
DIALOG(R)File 275: Gale Group Computer DB(TM)

Coordinate Win32 threads using manual-reset and auto-reset events. (Tutorial)
Richter, Jeffrey
Microsoft Systems Journal , v8 , n10 , p35(16)
Oct , 1993
Document Type: Tutorial
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 8954 **Line Count:** 00689

INumReaders variable into a block of shared **memory**. The easiest way to do this would be to use **memory**-mapped files.

The last thing to do would be to pay more attention to cleaning up when the user terminates one of the processes. You...

...itself with this possibility is because everything's in one process. If the global variable becomes invalid, it doesn't matter because all of the **other threads** are **terminated** too.

OK, I'll admit that Bucket is contrived; its main purpose is to demonstrate the problem of synchronizing multiple readers and writers. It should...

...events behave more like mutexes and semaphores than manual-reset events. When a thread calls SetEvent to signal an event, the event stays signaled until **another thread waiting** for the event is awakened. Just before the waiting thread is resumed, the event is reset automatically by the system. This is similar to the...

...before WaitForSingleObject or WaitForMultipleObjects returns, the system automatically resets the mutex. Using an auto-reset event in this way has the effect of allowing only **one thread** waiting for

the event to **resume** execution. Any other waiting threads are left suspended. This also means that you have no control over which of the suspended threads will resume execution...

20/3,K/3 (Item 3 from file: 275)
DIALOG(R)File 275: Gale Group Computer DB(TM)

Windows questions and answers: trapping floating-point exceptions; scrolling graphics. (solutions to problems in programming the Microsoft Windows graphical user interface and Windows-compliant applications) (Technical)
Dehoney, Michael; Vezzoli, Gian Camillo; Grover, Shuchi; Arra, Shravan; Bonneau, Paul
Windows-DOS Developer's Journal , v3 , n5 , p53(8)
May , 1992
Document Type: Technical
Language: ENGLISH **Record Type:** FULLTEXT
Word Count: 2451 **Line Count:** 00184

instruction. If an application clears a mask bit when the corresponding status bit is set, the ERROR signal is immediately asserted.

When Windows switches from **one task** to **another**, it stores the FPU's control word in the outgoing task's PDB (program data base) at offset 20. The control word is then set...
...if win87em.dll provided exception notification to an applicaiton, it would be required to do so by using Post mesage() and the application would not **learn** that an **exception** had occurred until the next call to GetMessage() or PeekMessage(). Instead of **waiting** for notification of an exception, your application could poll the FPU from within its floating-point routines to see if any of the status exception...

20/3,K/4 (Item 4 from file: 275)
DIALOG(R)File 275: Gale Group Computer DB(TM)

OS/2 power under DOS. (using the Phar Lap 286/DOS-Extender to write an OS/2 application that runs under MS-DOS) (Tutorial)
Eddon, Guy
Windows-DOS Developer's Journal , v3 , n2 , p49(4)
Feb , 1992
Document Type: Tutorial
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 1109 **Line Count:** 00088

to achieve multitasking, multithreading, and 16 megabytes of program code and data under DOS. The same programs could have been written in assembler.

Listing 1, **MEMORY**.C, uses the standard **memory** allocation function malloc(). Normally, malzoc() cannot allocate more than 640kb of **memory**. Using Phar Lap's 286

DOS-Extender, however, it can allocate up to 16 megabytes of free **memory**. **MEMORY**.C can be compiled and run under DOS by anyone with Microsoft C5.1 or 6.0 installed with OS/2 libraries and Phar Lap...
...an example of a multithreading program with a critical section. It uses the OS/2 function DosCreateThread() to start two threads which execute concurrently. The **first thread continuously** prints "1" while the **second thread** prints "2," so you **end** up seeing a mixture of the two. A critical section stops both threads and begins to beep. Normally a critical section might consist of a...

20/3,K/8 (Item 8 from file: 275)
DIALOG(R)File 275: Gale Group Computer DB(TM)

The time slice: What is it? What effect does it have on a user process? How is it different

from a subslice?
Means, Bill
DG Review , v9 , n10 , p14(4)
April , 1989
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 2314 **Line Count:** 00168

it will keep the CPU. Yes, I might have a two second time slice, but it might take me ten minutes to get that much **CPU time**, because all of the interactive processes will be continuously scheduled ahead of me. It's important to remember that AOS/VS will run its scheduler...
...VS. given that the two processes are at the same priority and the tasks within each process are at the same priority:
This behavior will **continue** until **one process** or the **other** either blocks or **terminates**. The two processes will continue to compete equally with each other for the CPU as long as each of them continues to use it for...

20/3,K/10 (Item 1 from file: 148)
DIALOG(R)File 148: Gale Group Trade & Industry DB

Long-memory inflation uncertainty: evidence from the term structure of interest rates. (includes related article commenting on the research)(Inflation Uncertainty)
Backus, David K.; Zin, Stanley E.; Jefferis, Richard H., Jr.
Journal of Money, Credit & Banking , v25 , n3 , p681(28)
August , 1993
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 5505 **Line Count:** 00433

long yields, even for yields on bonds with maturities up to ten years. We attempt to reconcile these two properties using the so-called fractional **difference process** introduced into economics by Granger and Joyeux (1980). With this process the variability of long yields approaches zero, but at a rate slower than exponential...
...the ability of the fractional difference model to mimic some of the features of short-term interest rates, inflation, and money growth. With the possible **exception** of money growth, we **find** that the fractional difference model performs well relative to stationary ARMA or random walk models. Thus the model is able to reproduce important features of both the long **end** of the yield curve and the high-order autocorrelations of short rates and inflation. We speculate that the fractional short rate cum inflation process might...
...by Campbell (1986) and Hansen and Jagannathan (1991), is to start with an equilibrium price measure, or intertemporal marginal rate of substitution: given a stochastic **process** for **one**-period state-contingent claims prices, we construct prices of risk-free bonds of different maturities. A second approach, common in finance, is to start with...

20/3,K/13 (Item 2 from file: 15)
DIALOG(R)File 15: ABI/Inform(R)

Seeing into the world of fiber optics for security
De Lia, Robert
Security Management Supplement pp: 7A-11A
Mar 1993
Word Count: 3196

rays follow longer paths than others. The lowest order mode, the axial ray traveling down the center of the fiber without reflecting, arrives at the **end** of the fiber before the higher order modes that strike the core-to-cladding interface close to the critical angle, therefore following longer paths. The...is called modal dispersion. Single-mode fiber is the most effective means of limiting model dispersion

since its core diameter is small enough that the **fiber** propagates only **one** mode efficiently. However, today's graded-index multimode fibers also limit model dispersion and the results are comparable. This should be understood when considering **one fiber** over the **other** for use in CCTV systems. CABLE. Unlike glass, glass fiber--because of its purity and size—is flexible and does not have the same characteristics...
...burial, general purpose, aerial, multiconductor, and plenum rated. Typically, the internal glass fiber is the same for all types of fiber cable with some small **exceptions**.
RECEIVER. The **detector** in the fiber-optic system converts the optical signal into an electrical signal compatible with conventional equipment and communications networks. A good signal detector responds...

Q' A: Win32. (question-and-answer) (Column)
Richter, Jeffrey
Microsoft Systems Journal , v9 , n6 . p79(3)
June . 1994
Document Type: Column
Language: ENGLISH **Record Type:** FULLTEXT
Word Count: 2730 **Line Count:** 00269

recursive function several times can overflow the available stack space. When this happens. a message box is presented to the user and the application is **terminated** by the system. I know that Win32 threads have a much larger stack (usually 1MB) but my recursive function could still overflow the stack if...
...the Win32 Programmer's Reference.)

When I first started thinking about this problem, I thought that it would be impossible to handle a stack overflow **exception**, because when the system notifies your thread of any **exception**, the system needs to evaluate an **exception** filter expression that you supply. Frequently, this **exception** filter expression calls another function; this function will require additional stack space. I thought that since a stack overflow **exception** occurred, there is no way that the system would be able to successfully call another function.

However. as you can see by Figure 3 above, the system raises the stack overflow **exception** just as it's committing the final page of physical storage. This means that there is actually a whole page of storage available for the...
...expression function. As long as your filter expression function uses less than a page's-worth of local variables, there is no problem.

If the **thread continues** to use the stack after the stack overflow **exception** is raised, all of the memory in the page at 0x08001000 will be used and the thread will attempt to access memory in the page starting at 0x08000000. When the thread attempts to access this reserved (noncommitted) memory, the system raises an access violation **exception**. If an access violation **exception** is raised while attempting to access the stack, the thread is in very deep trouble. You can't write code that can deal with this...
...the user, and terminate the process--not just the thread, the whole process.

So as you can see, when the system raises a stack overflow **exception**, it's really just a warning that an access violation **exception** is just around the corner and that you should take action now. To recover from this situation gracefully may mean terminating the thread, presenting a message box to the user, **terminating** the **process**. or any other action as long as it doesn't require much more stack space.

Stepping up to 32 bits: Chicago's process, thread, and memory management. (Tutorial)
Pietrek, Matt

Microsoft Systems Journal , v9 , n8 , p27(13)
August , 1994
Document Type: Tutorial
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 11310 **Line Count:** 00882

a whole megabyte of RAM for each thread stack. Instead, Chicago uses a mechanism known as a guard page to know when to commit additional **memory** in the stack's address range. Guard pages are an example of structured exception handling, which I'll talk about in my next article.

Another...This causes the thread to block and not waste any CPU cycles. When the sorting thread completes its work, it calls SetEvent, causing the other **thread** to wake up and **resume** execution. Not only has the CPU been used efficiently, but you've also avoided concurrency problems by preventing a thread from using data that may...hasn't been claimed by the maximum number of threads already. the wait function simply bumps up the usage count of the semaphore and the **thread continues**. But if the semaphore is already maxed out, the thread that called the **wait** function will block until some **other thread** releases its claim to the semaphore. A **thread** indicates that it's finished using a semaphore by passing the semaphore handle to ReleaseSemaphore.

The third type of synchronization object is the mutex. The term mutex is a contraction of the more familiar mutual exclusion. A program or set of programs uses a mutex when it wants only one **thread** at a time to access a resource or section of code. If one **thread** is using the resource, other **threads** are excluded from using it. One way to view a mutex is as a semaphore with a usage count of one. Using a mutex is very similar to using a semaphore. Each of the create, open, and release semaphore functions have mutex counterparts. When a **thread** needs to acquire a mutex, it calls one of the functions in the WaitForXxx family.

The fourth form of Win32 synchronization happens with critical sections. Unlike the other types of synchronization objects, critical sections can only be used by **threads** within the same **process**. Critical sections are there to prevent multiple threads from executing through the same section of code simultaneously. Compared to the other synchronization mechanisms, critical sections...to the header address, a program can do some additional lookups to find the address of the code, data, and resources for that module in **memory**.

In 16-bit Windows, the difference between an HMODULE and an HINSTANCE is not obvious but they really are different. An HINSTANCE in 16-bit...

26/3.K/5 (Item 5 from file: 275)
DIALOG(R)File 275: Gale Group Computer DB(TM)

Clearer, more comprehensive error processing with Win32 structured exception handling. (Tutorial)
Goodman, Kevin
Microsoft Systems Journal , v9 , n1 , p29(10)
Jan , 1994
Document Type: Tutorial
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 5242 **Line Count:** 00449

a few more uses for structured exception handling.

* Protecting critical resources. If you haven't implemented structured exceptions and an exception happens, your program will **terminate** without closing files handles, synchronization objects, or freeing memory. With structured exception handling, you can gurantee that your cleanup functions will be called.

* Protecting precious...the MessageBox in the exception handler will execute. Now, the difference between this exception handler and the system exception handler is that this process can **continue** executing. It will not be terminated like a process with an unhandled exception. Figure 2 lists all the possible hardware exceptions that can be generated...

...other details. Both hardware and software exceptions use this structure.

The CONTEXT structure describes the hardware state (registers and so on) at the time the **exception** occurred. This structure differs depending upon which machine your code is executing on.

The kernel then passes you a pointer to both of these structures in the **EXCEPTION** [underscore]POINTERS structure (also shown in Figure 3). **EXCEPTION** [underscore]RECORD and the three flavors of CONTEXT (Intel[R],MIPS, and Alpha AXP) are all defined in WINNT.H. Using the information provided in **EXCEPTION**[underscore]RECORD, you can decide if you want to handle the **exception** or not.

If an **exception** occurs within a try block, the **exception** filter gets evaluated. The **exception** filter is a parameter to the **exception** handler. Like any other parameter to a function, the **exception** filter can either be another function, a constant, or an expression that must evaluate to one of three options.

First your filter can "accept" the **exception** by evaluating to a 1 (**EXCEPTION**[underscore]EXECUTE[underscore]HANDLER). This instructs the dispatcher to call your **exception** handler. There are other ways to evalute to a 1 besides passing in **EXCEPTION** [underscore]EXECUTE[underscore]HANDLER. For example, the GetExceptionCode function retrieves the **exception** that caused the execution of the except block. GetExceptionCode is an intrinsic function only available to you in the filter or handler portion of a...

Convert C programs into multithreaded applications. (Tutorial)
Volkman, Victor R.
C Users Journal , v11 , n4 , p87(10)
April , 1993
Document Type: Tutorial
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 5016 **Line Count:** 00402

WAITING, ST...task function. **Alternately**, a **task** can simply return immediately to **terminate** itself. However, CTask does not free any resources still allocated at the time of termination. Thus, Wagner recommends kill...
...task only for fatal-**error** handling. Furthermore, if the task was executing a DOS function, then killing it explicitly may cause DOS to become unstable. The best practice is to...
...task function suspends a task for an indefinite period of time. Although a task may **suspend** itself, it must then rely on **another task** to call start...
...task. If all existing tasks are in the stopped state, the scheduler loops with interrupts enabled patiently waiting for a **task** to be **restarted**. The t...
...allows a task to put itself to sleep in a delayed state for a specified period. A sleeping task in the delayed state can be **resumed** by another **task** calling wake...
...**task** function also **resumes** a **task** waiting on any other event as well.

Additionally, CTask allows you to register save and restore functions via the set...
...funcs call. The save function is called just before a **task** is deactivated and the restore function is called just before scheduling it. These functions are ideally suited for maintaining the context of shared hardware resources...code which accesses shared program resources and must be executed without interruption. If a critical section were interrupted, it would face potential corruption of the **resources** by competing tasks. The "semaphore" is a special type of variable used to track mutual exclusion of processes. Specifically, CTask provides two different constructs called...

Writing a Windows debugger. (Tutorial)
Pietrek, Matt
Windows-DOS Developer's Journal , v3 , n6 , p6(7)
June , 1992
Document Type: Tutorial

Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 4812 **Line Count:** 00371

of the LOAD structure are interchanged.

If you execute an INT 21h function 4Bh with the AL set to 01h, DOS loads the process into **memory**, but does not jump to its entry point. Windows does not supply a similar function; calling WinExec() or LoadModule() starts up the new task with...
...a problem? Typcially, the majority of an application's code is invoked as a result of receiving a message. Suppose a debugger user sets a **breakpoint** somewhere in the message processing code. When the debuggee hits that **breakpoint**, the debuggee stops in the middle of its message processing. As a result. the Windows messaging system is never told that the message has been...still based on the "one message at a time" model. If you were to deadlock in the above scenario under OS/2, all message processing **threads** would be **suspended**, but **other** non-PM **threads** would **continue** to run.

Solutions

The high-powered Windows debugger currently available (TDW, CVW, and Mutiscope) all use a DLL from Microsoft called windebug.dll (QuickC for...
...dll provides a layer on top of some low-level undocumented Windows functionality. A windebug.dll user issues high-level commands such as "Load a **process**," "Step the **process**," or "Convert this logical address to a physical address." In short, windebug.dll does a lot of the hard work for you. The WINDEBUG API...
...notification handling. WINDEBUG shields the debugger from having to know abut interrupts by taking responsibility for installing the appropriate interrupt handlers (INT 1, INT 3, **exception** 13, and so on). As interrupts occur, WINDEBUG packages them up and treats them as just another event in the execution of the program. TOOLHELP...
...via a callback function. It is up to you to write the code for such things as obtaining the register values, determining what the current **task** is, and switching to the debugger's **task**.

The other significant issue that WINDEBUG shields you from is notification handling. When "significant" events occur in Windows, it optionally calls a notification handler. Significant events include a **task** starting up, a DLL being loaded, a request to output a debug string, a request for an input character ("Abort. Retry, Ignore"), and so on...unnaturally because the debugger kills it). Figure 2 shows the pseudocode for single-stepping or running the debuggee.

Memory Reading/Writing and Breakpoints

Accessing the **memory** of the debuggee process under Windows is a bit more difficult than under DOS. The first problem is writing to a code segment in protected...

Techniques for debugging multithread OS-2 programs with CodeView 2.2. (technical)
Petzold. Charles
Microsoft Systems Journal , v3 , n5 , p21(9)
Sept , 1988
Document Type: technical
Language: ENGLISH **Record Type:** FULLTEXT: ABSTRACT
Word Count: 3574 **Line Count:** 00274

001> BP CheckKey 001> G

The program creates the four threads, and they run normally, displaying their thread IDs in random locations on the screen.

Suspend the program by pressing the spacebar. Because you've set a breakpoint on the CheckKey function, pressing any key will cause CodeView to break at...you set a breakpoint for a particular thread.

The Go Command

We've used the Go command in the exercises involving freezing and unfreezing and **breakpoints**, and it's been working as we might expect. The Go command simply allows the program to run; control returns to CodeView when a **breakpoint** is encountered. The only oddity encountered was in

the freezing and unfreezing exercise when we froze all the threads and discovered that the Go command...

...last column in Figure 8, the Go command always temporarily unfreezes a particular thread if the thread is frozen. When control returns to CodeView, it **restores** the status of the **thread** to its former state. When using the thread-specific versions of the Go command (|nG, |.G, or |#G), CodeView temporarily freezes all threads except the...

...threads are prevented from running.

Let's run through a short exercise to see how this works. Load THREADS into CodeView: CVP THREADS

Set a **breakpoint** at the function WriteChar that applies only to thread 5 and Go: 001>|5BP WriteChar 001>G

CodeView will break when thread 5 reaches the...

...breaks: 005>G

Now try a Go command that applies only to thread 5: 005>|5G

In this case, only thread 5 runs. CodeView temporarily **freezes** all **other threads** before letting the program run. Clear the **breakpoint** and Go: 005>BC* 005>G

End the program by using the Escape key and exit CodeView: 005>Q

The Go command executed from the menu or the F5 key works the same as G entered in the dialog window. However, when a **breakpoint** occurs, it will not be immediately apparent which thread caused the break. For this reason, it's often preferable to use the G command in the dialog window when debugging multithread programs.

Breakpoints and

Singlestepping

Before proceeding, it will be helpful to review the debugging facilities supported by the 80286 microprocessor. A debugger can set a **breakpoint** at an assembly language instruction by replacing the first byte of the instruction with CCH, which is an Interrupt 3 instruction. When the program executes...

...by the debugger. DosPTrace is documented in the Programmer's Reference manual included in the Microsoft OS/2 Programmer's Toolkit.

When you set a **breakpoint** with the BP command, CodeView sets the **breakpoint** interrupt in your code, which is obvious. When you trace code using the T (Trace), P (Program Step), and E (Execute) commands, you may think that CodeView uses the single-step facility, but that's not necessarily true. When you trace through source code statements, CodeView uses a **breakpoint** at the end of the group of assembly language instructions that constitute a particular statement. When using the P command to trace through assembly language...command entered by itself.

If you ever find that the current thread is not the thread you expect or want, you can change the current **thread** with the CodeView Select command [is approx.]nS. This and other variations of the Select command are shown in Figure 11.

As you've seen...

Multiple tasks. (multitasking with OS-2)

Armbrust, Steven; Forgeron, Ted
PC Tech Journal , v5 , n11 , p90(9)
Nov , 1987
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 6693 **Line Count:** 00517

the regular class to ensure that all the threads in the class get a chance to run. Associated with the regular class is a maximum **wait** value (specified in units of seconds by the MAXWAIT entry in ...a semaphore can represent ownership of the resource. To do this, all threads must adhere to the convention that before any thread can access the **critical resource**, it must first request ownership of the semaphore by means of the DosSemRequest call. If it is not owned when DosSemRequest is issued, the issuing...

...of the same semaphore. If the threads choose to wait (instead of returning immediately), the ones that do not immediately gain access to the semaphore **continue** to wait until the owning **thread** releases the semaphore. When this happens, the waiting threads are dispatched again, and the highest-priority waiting thread becomes the new owner, even if a lower-priority thread has been waiting longer. The other **threads continue** waiting. If equal-priority **threads** are waiting, the **thread** that has been waiting the longest becomes the new owner.

Using a semaphore to represent ownership of some resource enables **threads** to read or write sensitive data areas without fear that other **threads** will interrupt them before they finish. Suppose that one **thread** gathers information about the positions of aircraft and records their x, y, and z coordinates in a database. Another **thread** reads the database and displays the aircraft positions on a screen. Without a semaphore guarding the database, the **thread** that writes the coordinates could write an airplane's x and y coordinates but then could be interrupted by the reading thread before updating the...

...thread can wait for another thread to clear the semaphore. The other call, DosMuxSemWait. is also similar to DosSemWait, but it enables a thread to **wait** until any one of a number of semaphores is cleared.

OS/2 defines two kinds of semaphores, system semaphores and RAM semaphores. System semaphores are...

OS/2 2.0: see how it runs. (testing applications on IBM's operating system)(includes related articles on OS/2 advantages over Microsoft Corp.'s DOS and Windows and on 32-bit applications)

Smith, Gina
PC-Computing , v5 , n10 , p190(7)
Oct , 1992
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 2497 **Line Count:** 00186

took 4 minutes and 27 seconds.

Excel for OS/2 Version 3.0

Once again, performance was virtually indistinguishable on the two platforms. The one **exception** was the scrolling test, which OS/2 2.0 took 3 minutes and 50 seconds to complete when there was an **idle task** in the background and which OS/2 1.3 took 3 minutes and 2 seconds to complete.

The Final Word

So should you switch from...

Company Names: International Business Machines Corp...

A phoenix from the ashes: IBM OS/2. (IBM's OS/2 2.0 operating system) (includes related article on the pros and cons of OS/2) (Special Report: Blueprint for the 1990s.)

Fawcett, Neil
Computer Weekly , p34(1)
Feb 20 , 1992
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 1127 **Line Count:** 00082

IBM plans to release its OS/2 2.0 operating system at the **end** of Mar 1992. A look at a beta version of the product indicates that it is very different from the earlier version. For example, IBM... ...memory still is a limitation in OS/2 2.0. The code runs well, but it can be sluggish if asked to do an intensive **task**. Also, OS/2 2.0 suffers from **program errors** and quirks, although the code is pretty stable.

Company Names: International Business Machines Corp...

32/3,K/4 (Item 4 from file: 275)
DIALOG(R)File 275: Gale Group Computer DB(TM)

IBM users should bring more down to desktop - and take objects seriously - Kutnick. (Meta Group's Dale Kutnick)
McGinn, Janice
Computergram International , n1783 , CGI10150009
Oct 15 , 1991
Language: ENGLISH **Record Type:** FULLTEXT
Word Count: 1096 **Line Count:** 00086

the emergence of technologies like object orientation and new communication infrastructures.
Mainframe-centric
While IBM continues to push a mainframe-centric environment – with some notable **exceptions** - Kutnick believes that it is incumbent upon users to integrate network-based sytems and to redeploy staff. The SystemView concept demands that data processing staff **stop** babysitting MVS on the mainframe, and start looking at what can be offloaded to the desktop. Large system prices are set to remain high, and...
...a distributed architecture. All office automation can be done on the desktop, and since most images are local, do we really need to manage and **process** them on a mainframe? Even IBM has recognised this with the January announcement of InfoPlus for the PS/2. Again, why stick to the mainframe...
...management from the desktop? Kutnick says that IBM Austin has NetView on the RS/6000, and he forecasts a 100 MIPS system by the year-**end**.
By Janice McGinn
Every other vendor manages the network from a workstation, but it's a big political issue with IBM. The same is true...
Company Names: International Business Machines Corp...

32/3,K/7 (Item 7 from file: 275)
DIALOG(R)File 275: Gale Group Computer DB(TM)

Tested mettle: the IBM RISC System/6000 Powerstation 320. (Hardware Review) (evaluation)
Wilson, David
UNIX Review , v8 , n10 , p103(11)
Oct , 1990
Document Type: evaluation
Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT
Word Count: 7308 **Line Count:** 00567

outperform other systems, but not by as wide a margin.
Report Card
The report card for the IBM RS/6000 is quite positive, with one **exception**: printed documentation. Installation, being both pre-loaded and IBM-installed, has to rate as "outstanding"; the only drawback is that an IBM technician comes out after the machine arrives, and will not be standing in the off ice **waiting** for the machine to arrive. Thus a delay of a day or two might be experienced while coordinating the installation **process**.
Documentation rates as either "outstanding" or poor"-for on-line and printed versions, respectively Although the InfoExplorer is a remarkable piece of work, we miss...
Company Names: International Business Machines Corp...

32/3,K/9 (Item 9 from file: 275)
DIALOG(R)File 275: Gale Group Computer DB(TM)

IBM announcements. (IBM details AIX 3 with hypertext for the RS/6000, offers RT migration and trade-ins, Applix Inc. offers AIX integrated office automation software) (product announcement)
Computergram International , n1387 , pCGI03190009
March 19 , 1990
Document Type: product announcement
Language: ENGLISH **Record Type:** FULLTEXT
Word Count: 1105 **Line Count:** 00091

view graphics with zoom pan capabilities, print information, or copy information to files for private use. The Hypertext Information Base contains documentation on general user **tasks**, system management **tasks**, communication **tasks** and commands. It can be implemented in a single-user workstation or a multi-user server environment. The Hypertext Information Base can reside on a...
...data from hardware and software in a standardised format. Errors are automatically recorded when they occur to enable analysis as part of the problem determination **process**. Concurrent error notification and reporting enables background notification of multiple services when an error is logged and enables the reporting of each error as it...
...logged. IBM also says that the system dump utility now includes the traditional kernel data plus some other data. The dump can be triggered by **exception** and formatted in either batch or interactive mode through the crash command. A dump can be initiated remotely. The System Event Performance Trace has a menu interface for starting and **stopping** the trace and generating trace reports. **Process** names are printed, elapsed time is reported between events. and the output is column aligned. Remote Services consists mostly of user interface facilities and an...
Company Names: International Business Machines Corp...